

Informe Proyecto Final

Proyecto de Detección de Frutas con Swin Transformer

Integrantes: Diego Cabezas
Profesores de cátedra: Javier Ruiz del Solar
Profesores ayudante: Jonas Peñailillo P.
Fecha de entrega: 14 de Diciembre de 2023
Santiago, Chile

Índice de Contenidos

1. Introducción	1
2. Metodología	2
2.1. Contexto:	2
2.2. Desarrollo del proyecto	3
3. Resultados	11
4. Análisis de Resultados	17
5. Conclusiones	19
6. Anexo	20
7. Bibliografía	41

Índice de Figuras

1. Arquitectura del Swin Transformer [1]	2
2. Primera Prueba - Comparación del modelo mediante la predicción vs la máscara real	12
3. Segunda Prueba - Comparación del modelo mediante la predicción vs la máscara real	12
4. Tercera Prueba - Comparación del Modelo Mediante la Predicción vs la Máscara Real	13
5. Cuarta Prueba - Comparación del Modelo Mediante la Predicción vs la Máscara Real	13
6. Quinta Prueba - Comparación del Modelo Mediante la Predicción vs la Máscara Real	14
7. Imagen del Test Número 1 y la Predicción del Modelo	15
8. Imagen del Test Número 2 y la Predicción del Modelo	15
9. Imagen del Test Número 2 y la Predicción del Modelo	16

Índice de Tablas

1. Resultados de evaluación en COCO.	11
2. Resultados de evaluación de segmentación semántica.	11

1. Introducción

En el transcurso de este informe, se presentará un análisis detallado de los resultados y la metodología empleada durante el desarrollo del proyecto final del curso de Procesamiento Avanzado de Imágenes (EL7008-1). Los objetivos de este proyecto consisten en la implementación de la arquitectura Swin Transformer para la detección de frutas, entrenar el modelo con la base de datos MinneApple y evaluar el rendimiento en la detección de manzanas en imágenes. El fin del proyecto es que los estudiantes profundicen en el entendimiento y aplicación de redes pre-entrenadas, con un énfasis particular en la arquitectura de los Transformers.

En la siguiente sección se relatara el desarrollo del proyecto y la importancia de los Swin Transformers en la vision computacional. Además de los pasos realizados para la creación del modelo y experimentación de los datos.

En la sección de Resultados se mostraran diversas predicciones del modelo durante la ejecución del proyecto. Además, en el Análisis de Resultados se vera el rendimiento del modelo en las diversas escalas de tamaño que poseen los objetos, analizando como este se comporta en las métricas de detección y segmentación.

La sección final de este informe se dedicará a resumir los principales aprendizajes obtenidos a lo largo del proyecto. Además, se abordarán las dificultades encontradas. Y las conclusiones obtenidas con respecto al uso de swin-t transformer para la segmentación semántica de objetos.

2. Metodología

2.1. Contexto:

Swin Transformer es un modelo creado por un equipo de Microsoft, cuyo objetivo es funcionar como un backbone general para el uso de visión computacional en la solución de diversos tipos de problemas. Construye mapas de características jerárquicas combinando patches de imágenes, tal como se observa en la figura 1. La ventaja que posee este modelo es que tiene una complejidad lineal. Esto debido a que se realiza self attention sobre las ventanas locales. La principal desventaja del uso de ventanas es la perdida de información por la descomunicación entre ellas. La solución que propone este modelo es usar shifted windows, es decir, en la segunda capa de self attention las ventanas se desplazan de tal forma que están superpuestas entre si. Otorgando información relevante de la imagen global [1].

El usar global self attention sobre toda la imagen causa que el computo de los tokens produzca una complejidad cuadrática. En general este tipo de acercamiento tiene un alto costo para imágenes de alta resolución. Por lo que el uso de Swin Transformer permite acercarse a los problemas de visión computacional con un costo aceptable. Cabe mencionar que se desarrollaron cuatro tipos de modelos en el paper 'Swin Transforme: hierarchical vision transfomer using Shifted Window' [1]. Siendo estos los siguientes:

- Swin-T: modelo de bajo costo con 96 canales y con 2,2,6,2 capas.
- Swin-S modelo de bajo costo con 96 canales y con 2,2,18,2 capas.
- Swin-B: modelo de mediano costo con 128 canales y con 2,2,18,2 capas
- Swin-L modelo de alto costo con 196 canales y 2,2,18,2 capas.

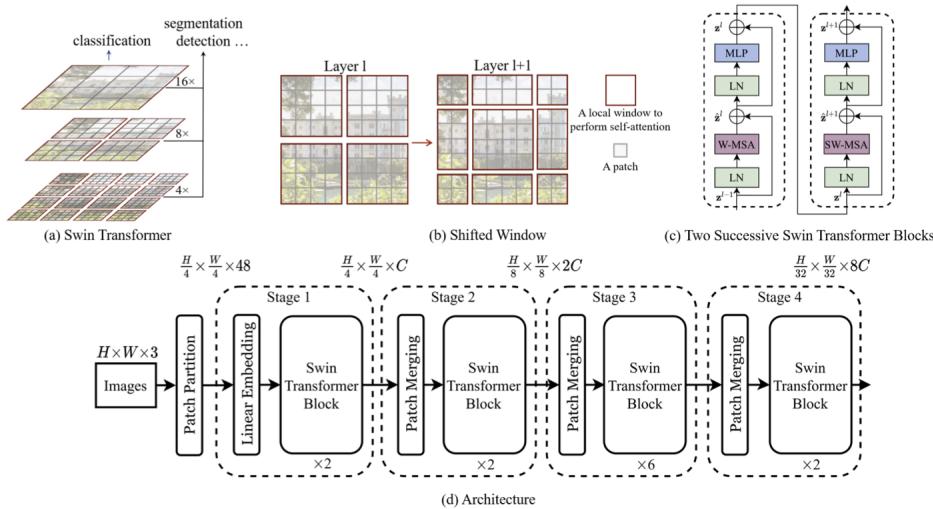


Figura 1: Arquitectura del Swin Transformer [1]

Se puede observar de la figura anterior que el Swin Transformer tiene su propio modulo de atención llamado W-MSA y SW-MSA los cuales toman en cuenta el desplazamiento de la ventana a la hora de

realizar el 'self attention'. Cabe destacar que los bloques de Swin Transformer provocan que el número de tokens se reduzca al aplicar 'patch merging' a medida que la red se profundiza. El linear embedding especificado por la letra C, permite limitar la complejidad computacional de los módulos de atención propia.

2.2. Desarrollo del proyecto

El proceso del desarrollo del proyecto empezó con una revisión bibliográfica del paper 'Swin Transformer: hierarchical vision transformer using Shifted Window'.

Con los conocimientos adquiridos se exploró el repositorio de Github de mmdetection [2], el cual cuenta con diversos modelos y backbones para el uso de semantic segmentation.

Se descargó el repositorio mmdetection y sus dependencias con el siguiente código:

```
!pip install -U openmim
!mim install mmengine
!mim install "mmcv>=2.0.0"
!git clone https://github.com/open-mmlab/mmdetection.git
%cd mmdetection
!pip install -v -e .
```

Mientras que el MinneApple Dataset se descargó mediante las siguientes líneas, con el fin de no ocupar la memoria del computador [5].

```
import os
import tarfile
import urllib.request

# URL del archivo comprimido
url = "https://conservancy.umn.edu/bitstream/handle/11299/206575/detection.tar.gz?sequence=2&
      → isAllowed=y"
file_name = "detection.tar.gz"

# Descargar el archivo
urllib.request.urlretrieve(url, file_name)
if not os.path.isfile(file_name):
    print("No se pudo descargar el archivo")
    exit()
# Descomprimir el archivo
with tarfile.open(file_name, "r:gz") as tar:
    print("descomprimiendo")

tar.extractall()
```

Al explorar el repositorio se encontró el modelo de swin-t con mask RCNN. Al elegir este como el modelo base deseado, se investigó cómo utilizar bases de datos propias para el entrenamiento de la red.

Por lo que se descubrió que para el uso del modelo es necesario la alteración de los archivos de configuración. Al contar con los archivos json de las anotaciones del Dataset MinneApple, se descubrió que

estos se encontraban en formato COCO por lo que no se tuvo que realizar alteraciones en estos para el funcionamiento del modelo.

El archivo de configuración contaba con el siguiente código:

```
_base_ = [
    '../_base_/models/mask-rcnn_r50_fpn.py',
    '../_base_/datasets/coco_instance.py',
    '../_base_/schedules/schedule_1x.py', '../_base_/default_runtime.py'
]

pretrained = 'https://github.com/SwinTransformer/storage/releases/download/v1.0.0/
    ↪ swin_tiny_patch4_window7_224.pth' # noqa

model = dict(
    type='MaskRCNN',
    backbone=dict(
        _delete_=True,
        type='SwinTransformer',
        embed_dims=96,
        depths=[2, 2, 6, 2],
        num_heads=[3, 6, 12, 24],
        window_size=7,
        mlp_ratio=4,
        qkv_bias=True,
        qk_scale=None,
        drop_rate=0.,
        attn_drop_rate=0.,
        drop_path_rate=0.2,
        patch_norm=True,
        out_indices=(0, 1, 2, 3),
        with_cp=False,
        convert_weights=True,
        init_cfg=dict(type='Pretrained', checkpoint=pretrained)),
    neck=dict(in_channels=[96, 192, 384, 768]))

# augmentation strategy originates from DETR / Sparse RCNN
train_pipeline = [
    dict(type='LoadImageFromFile', backend_args={{_base_.backend_args}}),
    dict(type='LoadAnnotations', with_bbox=True, with_mask=True),
    dict(type='RandomFlip', prob=0.5),
    dict(
        type='RandomChoice',
        transforms=[
            dict(
                type='RandomChoiceResize',
                scales=[(480, 1333), (512, 1333), (544, 1333), (576, 1333),
                    (608, 1333), (640, 1333), (672, 1333), (704, 1333),
                    (736, 1333), (768, 1333), (800, 1333)],
                keep_ratio=True)
        ]
]
```

```
[],
[
    dict(
        type='RandomChoiceResize',
        scales=[(400, 1333), (500, 1333), (600, 1333)],
        keep_ratio=True),
    dict(
        type='RandomCrop',
        crop_type='absolute_range',
        crop_size=(384, 600),
        allow_negative_crop=True),
    dict(
        type='RandomChoiceResize',
        scales=[(480, 1333), (512, 1333), (544, 1333),
               (576, 1333), (608, 1333), (640, 1333),
               (672, 1333), (704, 1333), (736, 1333),
               (768, 1333), (800, 1333)],
        keep_ratio=True)
]),
dict(type='PackDetInputs')
]
train_dataloader = dict(dataset=dict(pipeline=train_pipeline))

max_epochs = 36
train_cfg = dict(max_epochs=max_epochs)

# learning rate
param_scheduler = [
    dict(
        type='LinearLR', start_factor=0.001, by_epoch=False, begin=0,
        end=1000),
    dict(
        type='MultiStepLR',
        begin=0,
        end=max_epochs,
        by_epoch=True,
        milestones=[27, 33],
        gamma=0.1)
]

# optimizer
optim_wrapper = dict(
    type='OptimWrapper',
    paramwise_cfg=dict(
        custom_keys={
            'absolute_pos_embed': dict(decay_mult=0.),
            'relative_position_bias_table': dict(decay_mult=0.),
            'norm': dict(decay_mult=0.)
        },
        optimizer=dict(
            _delete_=True,
```

```

type='AdamW',
lr=0.0001,
betas=(0.9, 0.999),
weight_decay=0.05))

```

Al revisar el código anterior se percibió que el número de épocas es muy elevado para el uso de transformer, por lo que se decidió ocupar 4 épocas en comparación de las 36 propuestas. Además se observo que el modelo esta pre-entrenado con COCO el cual cuenta con la clase de manzanas.

Al observar las dependencias de los archivos en `__base__` se noto que el modelo esta confeccionado en el archivo 'mask-rcnn_r50_fpn.py'.

Mientras que los datasets y DataLoaders se crean en `__base__/datasets/coco_instance.py`. Tal como se menciono anteriormente como los json están en formato COCO, solo es necesario cambiar las dependencias de estos archivos de tal manera que ocupen las imágenes y anotaciones del MinneApple para la creación de los datasets. Es decir, cambiar las rutas de acceso de los datos en la creación del `train_dataloader` y `val_dataloader`.

```

dataset_type = 'CocoDataset'
data_root = '/content/detection/'
backend_args = None
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations', with_bbox=True, with_mask=True),
    dict(type='Resize', img_scale=(1280, 720), keep_ratio=True), # (1280,720), (640, 360), (320, 180)
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=8, pad_val=114),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels', 'gt_masks']),
]
test_pipeline = [
    dict(type='LoadImageFromFile', backend_args=backend_args),
    dict(type='Resize', scale=(1333, 800), keep_ratio=True),
    # If you don't have a gt annotation, delete the pipeline
    dict(type='LoadAnnotations', with_bbox=True, with_mask=True),
    dict(
        type='PackDetInputs',
        meta_keys=('img_id', 'img_path', 'ori_shape', 'img_shape',
                  'scale_factor'))
]
train_dataloader = dict(
    batch_size=2,
    num_workers=2,
    persistent_workers=True,
    sampler=dict(type='DefaultSampler', shuffle=True),
    batch_sampler=dict(type='AspectRatioBatchSampler'),
    dataset=dict(

```

```

type=dataset_type,
data_root=data_root,
ann_file=data_root + 'train/instances_train.json',
data_prefix=dict(img=data_root + 'train/images/'),
filter_cfg=dict(filter_empty_gt=True, min_size=32),
pipeline=train_pipeline,
backend_args=backend_args)
val_dataloader = dict(
    batch_size=1,
    num_workers=2,
    persistent_workers=True,
    drop_last=False,
    sampler=dict(type='DefaultSampler', shuffle=False),
    dataset=dict(
        type=dataset_type,
        data_root=data_root,
        ann_file=data_root + 'train/instances_val.json',
        data_prefix=dict(img=data_root + 'train/images/'),
        test_mode=True,
        pipeline=test_pipeline,
        backend_args=backend_args))
test_dataloader = val_dataloader

val_evaluator = dict(
    type='CocoMetric',
    ann_file=data_root + 'train/instances_val.json',
    metric=['bbox', 'segm'],
    format_only=False,
    outfile_prefix='./tutorial_exps/test',
    backend_args=backend_args)
test_evaluator = val_evaluator

#evaluation metrics
evaluation = dict(metric=['bbox'], save_best='bbox_mAP',
                  metric_items=['AR@100', 'AR@300', 'AR@1000', 'AR_s@1000', 'mAP', 'mAP_50', 'mAP_75',
                                'mAP_s', 'mAP_m', 'mAP_l'])

```

Donde se guardo los archivos json en la carpeta train del MinneApple. Siendo el data_root la carpeta donde se tiene descargado el dataset.

Cabe notar que el train_pipe y test_pipe realiza diversas transformaciones de data augmentation. A la vez que normaliza las imágenes con el fin de disminuir el costo de los cálculos del modelo.

Se guarda el archivo levemente modificado de la configuración config_swin2.py a '/content/mmdetection/configs/' y el archivo coco_instance_base.py a '/content/mmdetection/configs/base/datasets (Ambos archivos se encuentran en el repositorio de Github, link al final de esta sección).

Finalmente modiflico la configuración para que el modelo guardara los resultados cada dos épocas, además de especificar que solo existe la clase de manzanas y que la segmentación sea realizada con una paleta de colores determinada. Finalmente se inicializo el visualizer para observar las curvas de entrenamiento del

modelo.

```
from mmengine.runner import set_random_seed

# Modify dataset classes and color
cfg.metainfo = {
    'classes': ('apple', ),
    'palette': [
        (220, 20, 60),
    ]
}

# Modify num classes of the model in box head and mask head
cfg.model.roi_head.bbox_head.num_classes = 1
cfg.model.roi_head.mask_head.num_classes = 1
# Set up working dir to save files and logs.
cfg.work_dir = './tutorial_exps'
# We can set the evaluation interval to reduce the evaluation times
cfg.train_cfg.val_interval = 2
# We can set the checkpoint saving interval to reduce the storage cost
cfg.default_hooks.checkpoint.interval = 2
# Set seed to facilitate reproducing the result
cfg.seed = 0
set_random_seed(0, deterministic=False)
# We can also use tensorboard to log the training process
cfg.visualizer.vis_backends.append({'type': 'TensorboardVisBackend'})
#-----
config=f'/content/mmdetection/configs/swin/config_swin2.py'
with open(config, 'w') as f:
    f.write(cfg.pretty_text)
```

Con los cambios descritos anteriormente se construyo la configuración final del modelo, el cual esta descrito en la sección de Anexos del informe debido a la longitud de este.

Para el entrenamiento del modelo se ocupo la siguiente celda de código:

```
1 !python tools/train.py {config}
```

Mientras que para la visualización de las predicciones se ocupo el siguiente código:

```
1 cfg = Config.fromfile('/content/mmdetection/configs/swin/config_swin2.py')
2 import mmcv
3 from mmdet.apis import init_detector, inference_detector
4 img = mmcv.imread('/content/detection/train/images/20150921_132038_image286.png', channel_order='
    ↪ rgb')
5 checkpoint_file = '/content/mmdetection/tutorial_exps/epoch_4.pth'
6 model = init_detector(cfg, checkpoint_file, device='cpu')
7 new_result = inference_detector(model, img)
8 print(new_result)
9
```

```

10 from mmdet.registry import VISUALIZERS
11 # init visualizer(run the block only once in jupyter notebook)
12 visualizer = VISUALIZERS.build(model.cfg.visualizer)
13 # the dataset_meta is loaded from the checkpoint and
14 # then pass to the model in init_detector
15 visualizer.dataset_meta = model.dataset_meta
16
17 from mmengine.visualization import Visualizer
18 # get built visualizer
19 visualizer_now = Visualizer.get_current_instance()
20 # the dataset_meta is loaded from the checkpoint and
21 # then pass to the model in init_detector
22 visualizer_now.dataset_meta = model.dataset_meta
23 # show the results
24 visualizer_now.add_datasample(
25     'new_result',
26     img,
27     data_sample=new_result,
28     draw_gt=False,
29     wait_time=0,
30     out_file=None,
31     pred_score_thr=0.1
32 )
33 visualizer_now.show()

```

En la linea número 4 se puede cambiar el path para observar la predicción que realiza el modelo sobre la imagen deseada. Cabe notar que el *pred_score_thr* muestra solo las predicciones que superan cierto umbral, para la realización de este proyecto se decidió trabajar con un umbral de 0.1 con el fin de observar la mayor cantidad de predicciones aun cuando aparezcan falsos positivos.

Finalmente para comparar y visualizar la segmentación real de las imágenes, se ocupó la siguiente función, la cual plotea la máscara por encima de la figura original.

```

1 path='20150921_132038_image286.png'
2
3 im = mmcv.imread('/content/detection/train/images/' + path, channel_order='rgb')
4 masked = mmcv.imread('/content/detection/train/masks/' + path, channel_order='rgb')
5 def draw_mask(image, mask_generated):
6     masked_image = image.copy()
7
8     # Convert BGR image to RGB if not in RGB format
9     if image.shape[-1] == 3 and image.shape[-2] != 3:
10         masked_image = cv2.cvtColor(masked_image, cv2.COLOR_BGR2RGB)
11
12     # Create a green mask on the specified areas
13     mask = np.where(mask_generated.astype(int),
14                     np.array([0, 255, 0], dtype='uint8'),
15                     masked_image)
16
17     # Convert the masked image to uint8
18     masked_image = mask.astype(np.uint8)

```

```
19  
20 # Combine the original image and the masked image with a certain weight  
21 result = cv2.addWeighted(image, 0.3, masked_image, 0.7, 0)  
22  
23 return result  
24  
25 cv2_imshow(draw_mask(im,masked))
```

Github con el repositorio del proyecto en el siguiente link <https://github.com/Dieg0Bri/AvancesProyectoImagenes>.

3. Resultados

En la siguiente tabla se muestra los resultados de las métricas del modelo en la ultima época del entrenamiento.

Métrica	Valor
Bounding Box mAP	0.3310
Bounding Box mAP_50	0.7600
Bounding Box mAP_75	0.2240
Bounding Box mAP_s	0.3190
Bounding Box mAP_m	0.4840
Bounding Box mAP_l	-1.0000
Segmentation mAP	0.3420
Segmentation mAP_50	0.7670
Segmentation mAP_75	0.2460
Segmentation mAP_s	0.3190
Segmentation mAP_m	0.5320
Segmentation mAP_l	-1.0000

Tabla 1: Resultados de evaluación en COCO.

Al ver los resultados de la segmentación se obtiene el siguiente 'average recall' para los distintos umbrales del IoU. Con lo cual se confecciona la siguiente tabla:

Métrica	Valor
AR @[IoU=0.50:0.95, area=all, maxDets=100]	0.413
AR @[IoU=0.50:0.95, area=all, maxDets=300]	0.413
AR @[IoU=0.50:0.95, area=all, maxDets=1000]	0.413
AR @[IoU=0.50:0.95, area=small, maxDets=1000]	0.392
AR @[IoU=0.50:0.95, area=medium, maxDets=1000]	0.593
AR @[IoU=0.50:0.95, area=large, maxDets=1000]	-1.000

Tabla 2: Resultados de evaluación de segmentación semántica.

Con el entrenamiento terminado, se realiza las comparaciones entre la predicción hecha por el modelo y la mascara provista por el dataset de MinneApple. Se realizan 5 experimentos con lo cual se obtienen las siguientes figuras:



Figura 2: Primera Prueba - Comparación del modelo mediante la predicción vs la máscara real



Figura 3: Segunda Prueba - Comparación del modelo mediante la predicción vs la máscara real



Figura 4: Tercera Prueba - Comparación del Modelo Mediante la Predicción vs la Máscara Real

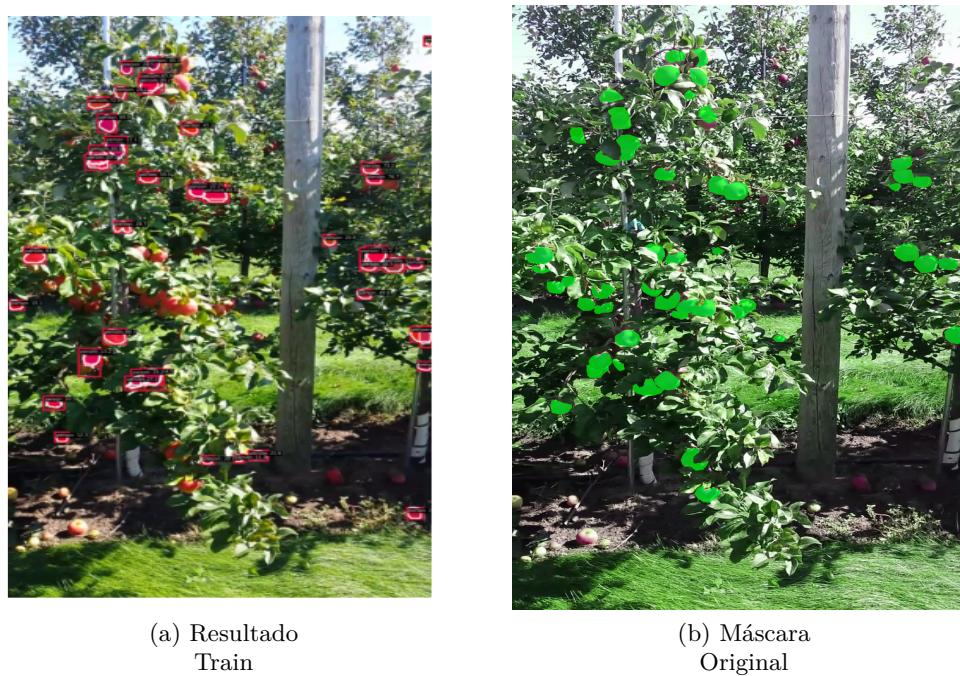


Figura 5: Cuarta Prueba - Comparación del Modelo Mediante la Predicción vs la Máscara Real

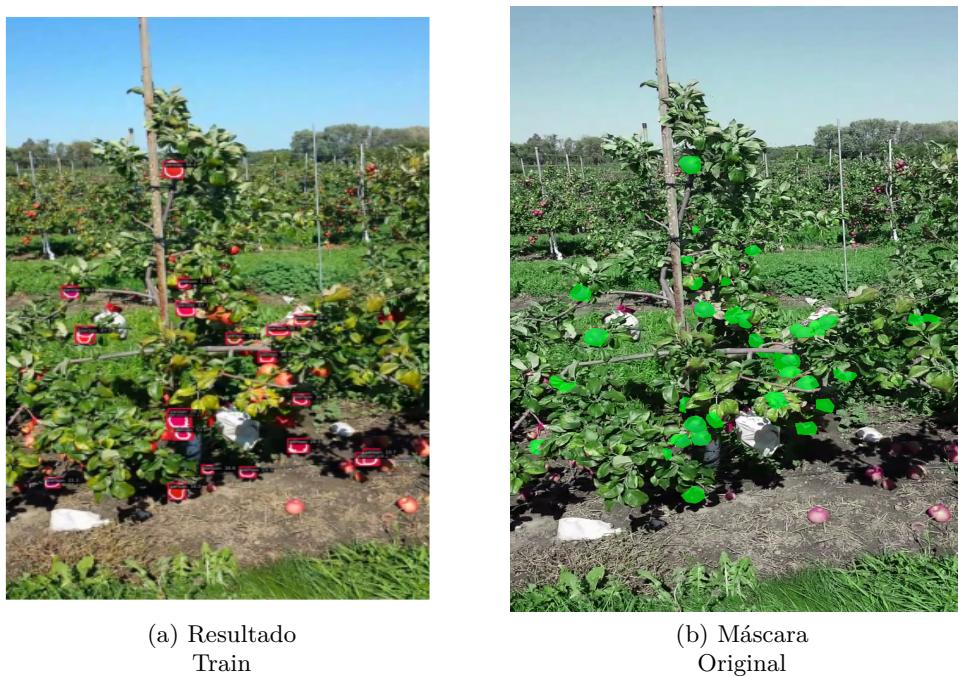


Figura 6: Quinta Prueba - Comparación del Modelo Mediante la Predicción vs la Máscara Real

Para ver como se enfrenta el modelo a datos nuevos se realiza las predicciones sobre las imágenes del conjunto de test. Con lo cual se puede distinguir visualmente el desempeño del modelo al observar las diferencias entre la imagen original y la segmentación semántica predicha. Las siguientes tres figuras describen esta diferencia.

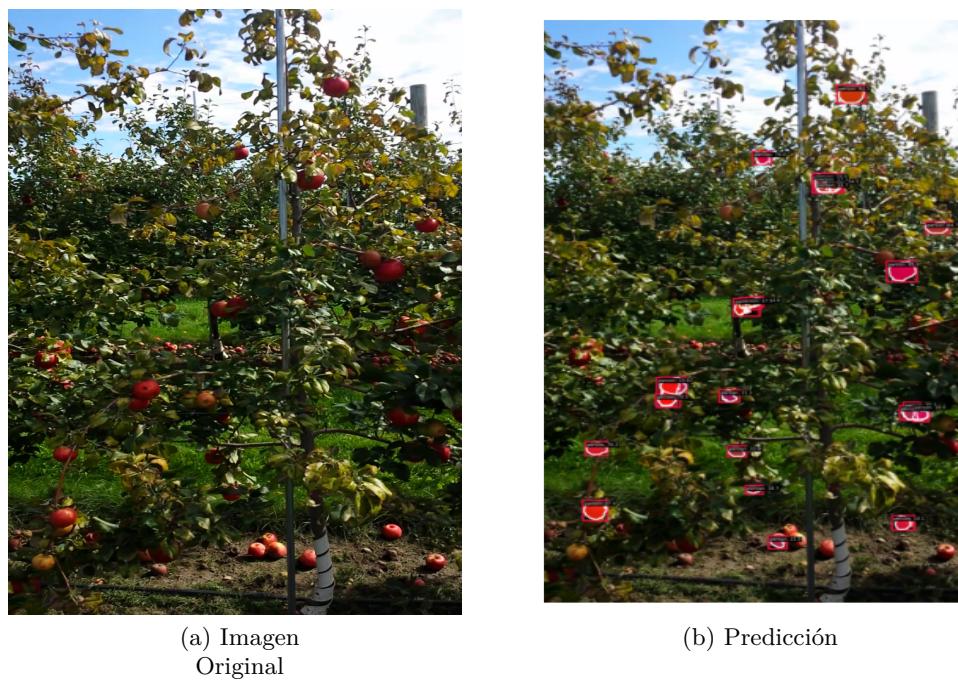


Figura 7: Imagen del Test Número 1 y la Predicción del Modelo

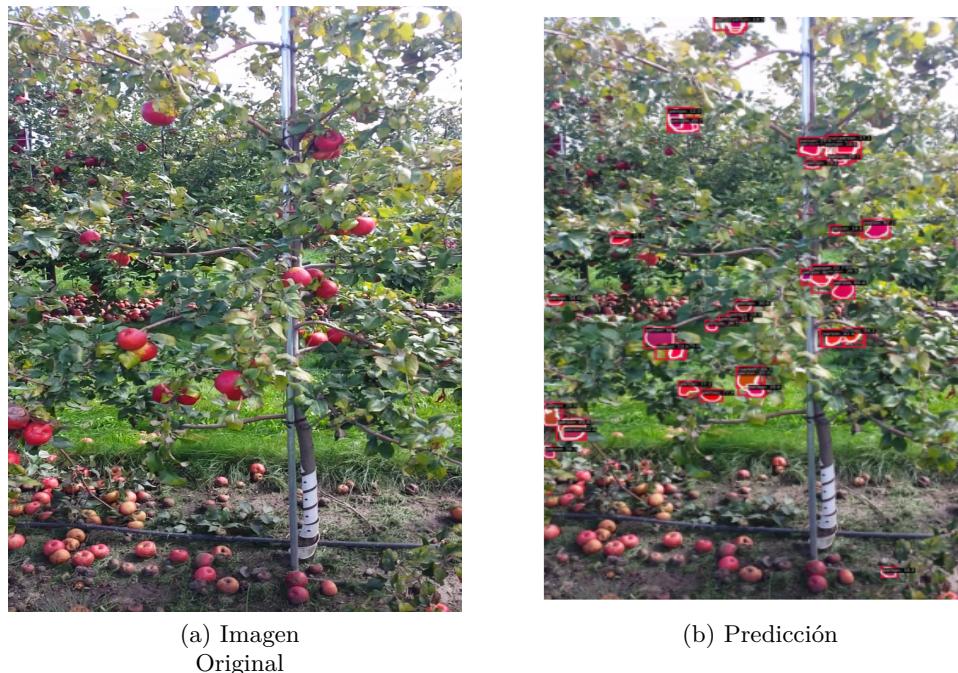


Figura 8: Imagen del Test Número 2 y la Predicción del Modelo

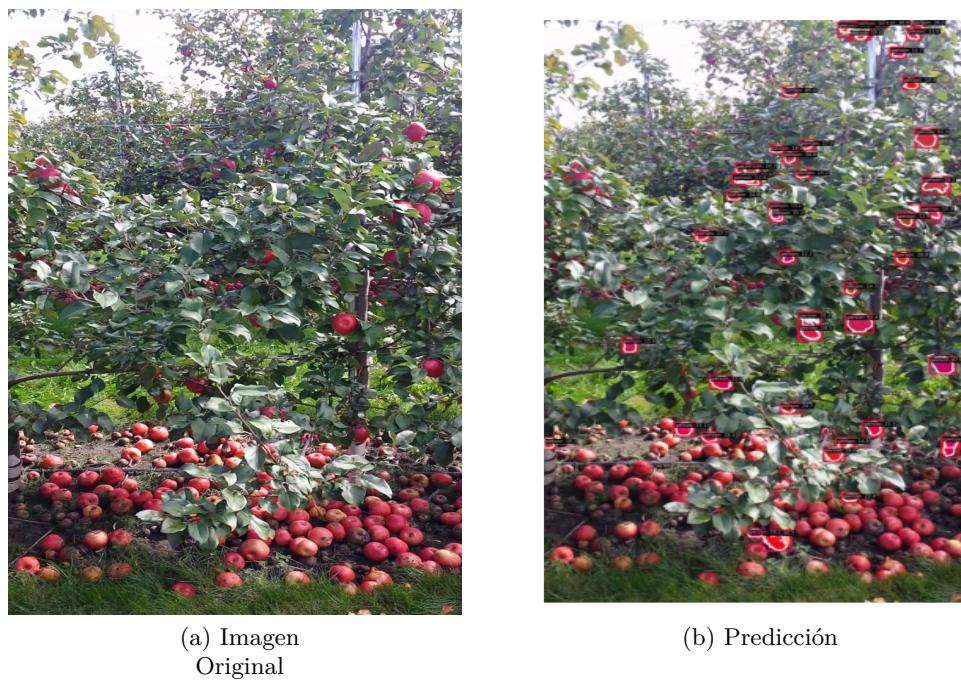


Figura 9: Imagen del Test Número 2 y la Predicción del Modelo

4. Análisis de Resultados

La tabla 1 proporciona resultados de métricas de evaluación para segmentación semántica. Al realizar un análisis de los resultados presentados se puede concluir lo siguiente:

La métrica mAP proporciona una evaluación general del rendimiento del Bounding Box. En este caso, la mAP es moderado (0.3310), lo que indica un rendimiento aceptable. Sin embargo, la mAP_75 es bajo (0.2240), lo que muestra que en comparación al mAP_50 el modelo genera una mejor calidad de las predicciones cuando el umbral de IoU es del 50 %.

La métrica mAP_s muestra un valor ligeramente menor que la mAP general. Esto indica que al modelo le dificulta la detección de objetos más pequeños.

La mAP específica para objetos medianos es más alto en comparación con el mAP general y el mAP para objetos pequeños. Esto sugiere un mejor rendimiento en la detección de objetos de tamaño mediano.

Un valor de -1.0000 para la mAP de objetos grandes indica que existe un problema en la detección o evaluación. Esto puede ser debido a dos opciones, o el tamaño de las ventanas en conjunto a los patches impide la identificación de los objetos de tamaño elevado o el conjunto de validación y test no posee objetos tan grandes, lo que impide su evaluación.

La mAP del rendimiento de la segmentación semántica mostró que en cuanto a las métricas específicas para umbrales (mAP_50 y mAP_75), existe un rendimiento moderado. Pero con un mejor desempeño del umbral del 50 %.

Similar a la métrica para objetos pequeños del bounding box, la mAP específica para segmentación muestra un rendimiento ligeramente inferior en comparación a los objetos de tamaño mediano. Obteniendo un 53.2 % en la precisión promedio de la segmentación semántica.

Tal como se menciono anteriormente se obtuvo un -1.0000 en el promedio de la precisión de la segmentación semántica para objetos grandes. Se espera que este error sea debido al modelo o debido al conjunto de evaluación.

Los resultados presentados en la tabla 2, están relacionados con las métricas de Recall. Las métricas de Recall promedio (AR) mide en qué proporción de todas las instancias de objetos es capaz de detectar el modelo. La variación de AR para los distintos tamaños de objetos muestra que el rendimiento es mejor en objetos de tamaño mediano en comparación a los objetos pequeños y grandes.

Lo mencionado anteriormente se puede observar visualmente al comparar las predicciones hechas en el entrenamiento versus las máscaras reales entregadas. Las cuales están descritas en las figuras 2, 3, 4, 5 y 6.

Con respecto a la figura 2, se puede observar que en comparación a la máscara, el modelo predijo menos frutas de lo esperado. Esto puede ser debido, a que el Dataset cuenta con menos manzanas verdes. Por lo que al modelo le cuesta su clasificación. También, puede ser por que el modelo esta pre entrenado con el COCO Dataset, y este podría contar con una menor cantidad de manzanas inmaduras.

Esto también es reflejado en la figura 3 donde se realizan tan solo dos detecciones. Además se visualiza claramente la dificultad del modelo de detectar objetos pequeños al observar que no se realizó ninguna segmentación de los arboles de fruta al fondo de la imagen. La dificultad puede ser debido al tamaño de la ventana del Swin Transformer, se esperaría que cambiar el tamaño de ventana permita detectar de mejor manera estos objetos.

El alto desempeño en las métricas que describen los objetos de tamaño mediano se pueden observar en la figura 4, donde se detectó y segmentó de manera correcta casi todas las manzanas vistas en la máscara. Sin embargo, se observan un par de falsos positivos en la esquina inferior izquierda, los cuales están asociados al umbral provisto para la visualización de las imágenes. El usar un threshold de 10 % permite observar detecciones correctas que poseen un bajo score, sin embargo, incrementa la cantidad de falsos positivos vistos en la imagen.

Las imágenes 5 y 6, muestran resultados similares al anterior, los cuales poseen un alto nivel de aciertos en cuanto a la detección y segmentación. Cabe destacar que las máscaras provistas por el modelo solo toma en consideración las frutas que se encuentran en los arboles, es por esto que las predicciones no detectan en general las manzanas que se encuentran en el suelo aunque estas sean fácilmente distinguibles.

Al realizar las pruebas sobre el conjunto de testeo, el cual está descrito en las figuras 7, 8 y 9. Se puede observar que el modelo funciona de manera adecuada, al comparar la posición de las manzanas con la segmentación realizada. Cabe destacar que en la última figura el umbral de score se disminuyó a 0.05, con el fin de observar si el modelo era capaz de detectar las manzanas que se encontraban en el suelo. No obstante, se refleja en la imagen pocas detecciones de las frutas en el suelo. Siendo las frutas detectadas las que se encuentran cerca de la ramas del árbol. Este resultado está directamente relacionado con las máscaras provistas por el Dataset de MinneApple.

Una forma de mejorar el desempeño del algoritmo es agregándole fotos y máscaras de los objetos de mayor tamaño, entrenándolo en diversas escalas de ventanas. A la vez se podría agregar transformaciones a las imágenes de tal manera que disminuyan los efectos de la iluminación y el movimiento de la cámara.

5. Conclusiones

En cuanto a las conclusiones la experiencia sirvió para profundizar los aprendizajes obtenidos en clase, especialmente acerca del uso Transformers en problemas de visión computacional.

Con respecto al objetivo del proyecto, este fue cumplido ya que se implemento correctamente la arquitectura de Swin Transformers, además que se entreno con la base de datos de MinneApple, obteniendo un rendimiento adecuado para la tarea propuesta.

Está tarea provee un acercamiento al uso de las distintas redes ya creadas. Los principales aprendizajes se centran en la alteración de archivos, y la importancia de guardar la información del entrenamiento a lo largo de la red.

La principal fuente de error, fue con la creación de la configuración del modelo. Especialmente por las dependencias de archivos a lo largo de las diversas carpetas.

Visualmente la segmentación es adecuada, sin embargo, puede ser mejorada, al cambiar los parámetros como el número de épocas. A la vez se podría probar con modelos de mayor complejidad como es Swin-B o Swin-L. También cambiar los tamaños de las ventanas podría permitir mejorar la detección de objetos de mayor o menor tamaño. El ajuste de los parámetros es fundamental para la optimización de este modelo, por lo que se espera que al cambiar estos, se obtenga un mejor resultado para la problemática propuesta.

6. Anexo

Código de la configuración final:

```
1 auto_scale_lr = dict(base_batch_size=16, enable=False)
2 backend_args = None
3 classes = [
4     'apple',
5 ]
6 data_root = '/content/detection/'
7 dataset_type = 'CocoDataset'
8 default_hooks = dict(
9     checkpoint=dict(interval=2, type='CheckpointHook'),
10    logger=dict(interval=50, type='LoggerHook'),
11    param_scheduler=dict(type='ParamSchedulerHook'),
12    sampler_seed=dict(type='DistSamplerSeedHook'),
13    timer=dict(type='IterTimerHook'),
14    visualization=dict(type='DetVisualizationHook'))
15 default_scope = 'mmdet'
16 env_cfg = dict(
17     cudnn_benchmark=False,
18     dist_cfg=dict(backend='nccl'),
19     mp_cfg=dict(mp_start_method='fork', opencv_num_threads=0))
20 evaluation = dict(
21     metric=[
22         'bbox',
23     ],
24     metric_items=[
25         'AR@100',
26         'AR@300',
27         'AR@1000',
28         'AR_s@1000',
29         'mAP',
30         'mAP_50',
31         'mAP_75',
32         'mAP_s',
33         'mAP_m',
34         'mAP_l',
35     ],
36     save_best='bbox_mAP')
37 img_norm_cfg = dict(
38     mean=[
39         123.675,
40         116.28,
41         103.53,
42     ],
43     std=[
44         58.395,
45         57.12,
46         57.375,
47     ],
```

```
48     to_rgb=True)
49 launcher = 'none'
50 load_from = None
51 log_level = 'INFO'
52 log_processor = dict(by_epoch=True, type='LogProcessor', window_size=50)
53 max_epochs = 4
54 metainfo = dict(
55     classes=('apple', ), palette=[
56         (
57             220,
58             20,
59             60,
60         ),
61     ],
62 model = dict(
63     backbone=dict(
64         attn_drop_rate=0.0,
65         convert_weights=True,
66         depths=[
67             2,
68             2,
69             6,
70             2,
71         ],
72         drop_path_rate=0.2,
73         drop_rate=0.0,
74         embed_dims=96,
75         init_cfg=dict(
76             checkpoint=
77                 'https://github.com/SwinTransformer/storage/releases/download/v1.0.0/
78             ↪ swin_tiny_patch4_window7_224.pth',
79             type='Pretrained'),
80         mlp_ratio=4,
81         num_heads=[
82             3,
83             6,
84             12,
85             24,
86         ],
87         out_indices=(
88             0,
89             1,
90             2,
91             3,
92         ),
93         patch_norm=True,
94         qk_scale=None,
95         qkv_bias=True,
96         type='SwinTransformer',
97         window_size=7,
98         with_cp=False),
```

```
98     data_preprocessor=dict(
99         bgr_to_rgb=True,
100        mean=[
101            123.675,
102            116.28,
103            103.53,
104        ],
105        pad_mask=True,
106        pad_size_divisor=32,
107        std=[
108            58.395,
109            57.12,
110            57.375,
111        ],
112        type='DetDataPreprocessor'),
113    neck=dict(
114        in_channels=[
115            96,
116            192,
117            384,
118            768,
119        ],
120        num_outs=5,
121        out_channels=256,
122        type='FPN'),
123    roi_head=dict(
124        bbox_head=dict(
125            bbox_coder=dict(
126                target_means=[
127                    0.0,
128                    0.0,
129                    0.0,
130                    0.0,
131                ],
132                target_stds=[
133                    0.1,
134                    0.1,
135                    0.2,
136                    0.2,
137                ],
138                type='DeltaXYWHBoxCoder'),
139            fc_out_channels=1024,
140            in_channels=256,
141            loss_bbox=dict(loss_weight=1.0, type='L1Loss'),
142            loss_cls=dict(
143                loss_weight=1.0, type='CrossEntropyLoss', use_sigmoid=False),
144            num_classes=1,
145            reg_class_agnostic=False,
146            roi_feat_size=7,
147            type='Shared2FCBBoxHead'),
148            bbox_roi_extractor=dict(
```

```
149     featmap_strides=[  
150         4,  
151         8,  
152         16,  
153         32,  
154     ],  
155     out_channels=256,  
156     roi_layer=dict(output_size=7, sampling_ratio=0, type='RoIAlign'),  
157     type='SingleRoIExtractor'),  
158     mask_head=dict(  
159         conv_out_channels=256,  
160         in_channels=256,  
161         loss_mask=dict(  
162             loss_weight=1.0, type='CrossEntropyLoss', use_mask=True),  
163             num_classes=1,  
164             num_convs=4,  
165             type='FCNMaskHead'),  
166     mask_roi_extractor=dict(  
167         featmap_strides=[  
168             4,  
169             8,  
170             16,  
171             32,  
172         ],  
173         out_channels=256,  
174         roi_layer=dict(output_size=14, sampling_ratio=0, type='RoIAlign'),  
175         type='SingleRoIExtractor'),  
176         type='StandardRoIHead'),  
177     rpn_head=dict(  
178         anchor_generator=dict(  
179             ratios=[  
180                 0.5,  
181                 1.0,  
182                 2.0,  
183             ],  
184             scales=[  
185                 8,  
186             ],  
187             strides=[  
188                 4,  
189                 8,  
190                 16,  
191                 32,  
192                 64,  
193             ],  
194             type='AnchorGenerator'),  
195         bbox_coder=dict(  
196             target_means=[  
197                 0.0,  
198                 0.0,  
199                 0.0,
```

```
200         0.0,
201     ],
202     target_stds=[
203         1.0,
204         1.0,
205         1.0,
206         1.0,
207     ],
208     type='DeltaXYWHBoxCoder'),
209     feat_channels=256,
210     in_channels=256,
211     loss_bbox=dict(loss_weight=1.0, type='L1Loss'),
212     loss_cls=dict(
213         loss_weight=1.0, type='CrossEntropyLoss', use_sigmoid=True),
214     type='RPNHead'),
215     test_cfg=dict(
216         rcnn=dict(
217             mask_thr_binary=0.5,
218             max_per_img=100,
219             nms=dict(iou_threshold=0.5, type='nms'),
220             score_thr=0.05),
221         rpn=dict(
222             max_per_img=1000,
223             min_bbox_size=0,
224             nms=dict(iou_threshold=0.7, type='nms'),
225             nms_pre=1000)),
226     train_cfg=dict(
227         rcnn=dict(
228             assigner=dict(
229                 ignore_iof_thr=-1,
230                 match_low_quality=True,
231                 min_pos_iou=0.5,
232                 neg_iou_thr=0.5,
233                 pos_iou_thr=0.5,
234                 type='MaxIoUAssigner'),
235             debug=False,
236             mask_size=28,
237             pos_weight=-1,
238             sampler=dict(
239                 add_gt_as_proposals=True,
240                 neg_pos_ub=-1,
241                 num=512,
242                 pos_fraction=0.25,
243                 type='RandomSampler')),
244         rpn=dict(
245             allowed_border=-1,
246             assigner=dict(
247                 ignore_iof_thr=-1,
248                 match_low_quality=True,
249                 min_pos_iou=0.3,
250                 neg_iou_thr=0.3,
```

```
251     pos_iou_thr=0.7,
252     type='MaxIoUAssigner'),
253     debug=False,
254     pos_weight=-1,
255     sampler=dict(
256         add_gt_as_proposals=False,
257         neg_pos_ub=-1,
258         num=256,
259         pos_fraction=0.5,
260         type='RandomSampler')),
261     rpn_proposal=dict(
262         max_per_img=1000,
263         min_bbox_size=0,
264         nms=dict(iou_threshold=0.7, type='nms'),
265         nms_pre=2000),
266     type='MaskRCNN')
267 optim_wrapper = dict(
268     optimizer=dict(
269         betas=(
270             0.9,
271             0.999,
272         ), lr=0.0001, type='AdamW', weight_decay=0.05),
273     paramwise_cfg=dict(
274         custom_keys=dict(
275             absolute_pos_embed=dict(decay_mult=0.0),
276             norm=dict(decay_mult=0.0),
277             relative_position_bias_table=dict(decay_mult=0.0))),
278         type='OptimWrapper')
279 param_scheduler = [
280     dict(
281         begin=0, by_epoch=False, end=1000, start_factor=0.001,
282         type='LinearLR'),
283     dict(
284         begin=0,
285         by_epoch=True,
286         end=4,
287         gamma=0.1,
288         milestones=[
289             27,
290             33,
291         ],
292         type='MultiStepLR'),
293 ]
294 pretrained = 'https://github.com/SwinTransformer/storage/releases/download/v1.0.0/
295     ↪ swin_tiny_patch4_window7_224.pth'
295 resume = False
296 seed = 0
297 test_cfg = dict(type='TestLoop')
298 test_dataloader = dict(
299     batch_size=1,
300     dataset=dict(
```

```
301     ann_file='/content/detection/train/instances_val.json',
302     backend_args=None,
303     data_prefix=dict(img='/content/detection/train/images/'),
304     data_root='/content/detection/',
305     pipeline=[
306         dict(backend_args=None, type='LoadImageFromFile'),
307         dict(keep_ratio=True, scale=(
308             1333,
309             800,
310         ), type='Resize'),
311         dict(type='LoadAnnotations', with_bbox=True, with_mask=True),
312         dict(
313             meta_keys=(
314                 'img_id',
315                 'img_path',
316                 'ori_shape',
317                 'img_shape',
318                 'scale_factor',
319             ),
320             type='PackDetInputs'),
321         ],
322         test_mode=True,
323         type='CocoDataset'),
324     drop_last=False,
325     num_workers=2,
326     persistent_workers=True,
327     sampler=dict(shuffle=False, type='DefaultSampler'))
328 test_evaluator = dict(
329     ann_file='/content/detection/train/instances_val.json',
330     backend_args=None,
331     format_only=False,
332     metric=[
333         'bbox',
334         'segm',
335     ],
336     outfile_prefix='./tutorial_exps/test',
337     type='CocoMetric')
338 test_pipeline = [
339     dict(backend_args=None, type='LoadImageFromFile'),
340     dict(keep_ratio=True, scale=(
341         1333,
342         800,
343     ), type='Resize'),
344     dict(type='LoadAnnotations', with_bbox=True, with_mask=True),
345     dict(
346         meta_keys=(
347             'img_id',
348             'img_path',
349             'ori_shape',
350             'img_shape',
351             'scale_factor',
```



```
403             (
404                 704,
405                 1333,
406             ),
407             (
408                 736,
409                 1333,
410             ),
411             (
412                 768,
413                 1333,
414             ),
415             (
416                 800,
417                 1333,
418             ),
419         ],
420         type='RandomChoiceResize'),
421     ],
422     [
423         dict(
424             keep_ratio=True,
425             scales=[
426                 (
427                     400,
428                     1333,
429                 ),
430                 (
431                     500,
432                     1333,
433                 ),
434                 (
435                     600,
436                     1333,
437                 ),
438         ],
439         type='RandomChoiceResize'),
440     dict(
441         allow_negative_crop=True,
442         crop_size=(
443             384,
444             600,
445         ),
446         crop_type='absolute_range',
447         type='RandomCrop'),
448     dict(
449         keep_ratio=True,
450         scales=[
451             (
452                 480,
453                 1333,
```

```
454     ),
455     (
456         512,
457         1333,
458     ),
459     (
460         544,
461         1333,
462     ),
463     (
464         576,
465         1333,
466     ),
467     (
468         608,
469         1333,
470     ),
471     (
472         640,
473         1333,
474     ),
475     (
476         672,
477         1333,
478     ),
479     (
480         704,
481         1333,
482     ),
483     (
484         736,
485         1333,
486     ),
487     (
488         768,
489         1333,
490     ),
491     (
492         800,
493         1333,
494     ),
495     ],
496     type='RandomChoiceResize'),
497     ],
498     ],
499     type='RandomChoice'),
500     dict(type='PackDetInputs'),
501     ],
502     type='CocoDataset'),
503     num_workers=2,
504     persistent_workers=True,
```

```
505     sampler=dict(shuffle=True, type='DefaultSampler'))
506 train_pipeline = [
507     dict(type='LoadImageFromFile'),
508     dict(type='LoadAnnotations', with_bbox=True, with_mask=True),
509     dict(prob=0.5, type='RandomFlip'),
510     dict(
511         transforms=[
512             [
513                 dict(
514                     keep_ratio=True,
515                     scales=[
516                         (
517                             480,
518                             1333,
519                         ),
520                         (
521                             512,
522                             1333,
523                         ),
524                         (
525                             544,
526                             1333,
527                         ),
528                         (
529                             576,
530                             1333,
531                         ),
532                         (
533                             608,
534                             1333,
535                         ),
536                         (
537                             640,
538                             1333,
539                         ),
540                         (
541                             672,
542                             1333,
543                         ),
544                         (
545                             704,
546                             1333,
547                         ),
548                         (
549                             736,
550                             1333,
551                         ),
552                         (
553                             768,
554                             1333,
555                         ),
```

```
556             (
557                 800,
558                 1333,
559             ),
560         ],
561         type='RandomChoiceResize'),
562     ],
563     [
564         dict(
565             keep_ratio=True,
566             scales=[
567                 (
568                     400,
569                     1333,
570                 ),
571                 (
572                     500,
573                     1333,
574                 ),
575                 (
576                     600,
577                     1333,
578                 ),
579             ],
580             type='RandomChoiceResize'),
581         dict(
582             allow_negative_crop=True,
583             crop_size=(
584                 384,
585                 600,
586             ),
587             crop_type='absolute_range',
588             type='RandomCrop'),
589         dict(
590             keep_ratio=True,
591             scales=[
592                 (
593                     480,
594                     1333,
595                 ),
596                 (
597                     512,
598                     1333,
599                 ),
600                 (
601                     544,
602                     1333,
603                 ),
604                 (
605                     576,
606                     1333,
```

```
607         ),
608         (
609             608,
610             1333,
611         ),
612         (
613             640,
614             1333,
615         ),
616         (
617             672,
618             1333,
619         ),
620         (
621             704,
622             1333,
623         ),
624         (
625             736,
626             1333,
627         ),
628         (
629             768,
630             1333,
631         ),
632         (
633             800,
634             1333,
635         ),
636     ],
637     type='RandomChoiceResize'),
638     ],
639     ],
640     type='RandomChoice'),
641     dict(type='PackDetInputs'),
642 ]
643 val_cfg = dict(type='ValLoop')
644 val_dataloader = dict(
645     batch_size=1,
646     dataset=dict(
647         ann_file='/content/detection/train/instances_val.json',
648         backend_args=None,
649         data_prefix=dict(img='/content/detection/train/images/'),
650         data_root='/content/detection/',
651         pipeline=[
652             dict(backend_args=None, type='LoadImageFromFile'),
653             dict(keep_ratio=True, scale=(
654                 1333,
655                 800,
656             ), type='Resize'),
657             dict(type='LoadAnnotations', with_bbox=True, with_mask=True),
```

```

658     dict(
659         meta_keys=(
660             'img_id',
661             'img_path',
662             'ori_shape',
663             'img_shape',
664             'scale_factor',
665         ),
666         type='PackDetInputs'),
667     ],
668     test_mode=True,
669     type='CocoDataset'),
670     drop_last=False,
671     num_workers=2,
672     persistent_workers=True,
673     sampler=dict(shuffle=False, type='DefaultSampler'))
674 val_evaluator = dict(
675     ann_file='/content/detection/train/instances_val.json',
676     backend_args=None,
677     format_only=False,
678     metric=[
679         'bbox',
680         'segm',
681     ],
682     outfile_prefix='./tutorial_exps/test',
683     type='CocoMetric')
684 vis_backends = [
685     dict(type='LocalVisBackend'),
686 ]
687 visualizer = dict(
688     name='visualizer',
689     type='DetLocalVisualizer',
690     vis_backends=[
691         dict(type='LocalVisBackend'),
692         dict(type='TensorboardVisBackend'),
693     ])
694 work_dir = './tutorial_exps'

```

Código del proyecto en formato .py:

```

1 # -*- coding: utf-8 -*-
2 """ProyectoIMAGENES-locura.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1SUO5cAit79WdkAt-E01bPSD3uHbds-3j
8 """
9
10
11

```

```
12 import cv2
13 import numpy as np
14 from google.colab.patches import cv2_imshow
15 from numpy.linalg import norm
16 from sklearn.preprocessing import StandardScaler
17 import matplotlib.pyplot as plt
18
19 import logging
20 import numpy as np
21 import torch
22 from PIL import Image
23 from functools import lru_cache
24 from functools import partial
25 from itertools import repeat
26 from multiprocessing import Pool
27 from os import listdir
28 from os.path import splitext, isfile, join
29 from pathlib import Path
30 from torch.utils.data import Dataset
31 from tqdm import tqdm
32
33 """#DATASET IMPORT
34
35 Descargar el MineApple Dataset
36 """
37
38 import os
39 import tarfile
40 import urllib.request
41
42 # URL del archivo comprimido
43 url = "https://conservancy.umn.edu/bitstream/handle/11299/206575/detection.tar.gz?sequence=2&
44     ↪ isAllowed=y"
45 file_name = "detection.tar.gz"
46
47 # Descargar el archivo
48 urllib.request.urlretrieve(url, file_name)
49 if not os.path.isfile(file_name):
50     print("No se pudo descargar el archivo")
51     exit()
52
53 # Descomprimir el archivo
54 with tarfile.open(file_name, "r:gz") as tar:
55     print("descomprimiendo")
56
57     tar.extractall()
58
59 def load_image(filename):
60     ext = splitext(filename)[1]
61     if ext == '.npy':
62         return Image.fromarray(np.load(filename))
63     elif ext in ['.pt', '.pth']:
```

```
62     return Image.fromarray(torch.load(filename).numpy())
63 else:
64     return Image.open(filename)
65
66 img2=cv2.imread("/content/detection/train/images/20150919_174151_image301.png", cv2.
67   ↪ IMREAD_UNCHANGED)
67 img = load_image("/content/detection/train/masks/20150919_174151_image301.png")
68
69 cv2_imshow(img2)
70 plt.imshow(img)
71
72 """# MNIST GITHUB"""
73
74 !pip install -U openmim
75 !mim install mmengine
76 !mim install "mmcv>=2.0.0"
77
78 # Commented out IPython magic to ensure Python compatibility.
79 !git clone https://github.com/open-mmlab/mmdetection.git
80 # %cd mmdetection
81 !pip install -v -e .
82
83 """COnfiguracion del Swin-t
84
85 _base_ = [
86     '../_base_/models/mask-rcnn_r50_fpn.py',
87     '../_base_/datasets/coco_instance.py',
88     '../_base_/schedules/schedule_1x.py', '../_base_/default_runtime.py'
89 ]
90 pretrained = 'https://github.com/SwinTransformer/storage/releases/download/v1.0.0/
91   ↪ swin_tiny_patch4_window7_224.pth' # noqa
91 model = dict(
92     type='MaskRCNN',
93     backbone=dict(
94         _delete_=True,
95         type='SwinTransformer',
96         embed_dims=96,
97         depths=[2, 2, 6, 2],
98         num_heads=[3, 6, 12, 24],
99         window_size=7,
100        mlp_ratio=4,
101        qkv_bias=True,
102        qk_scale=None,
103        drop_rate=0.,
104        attn_drop_rate=0.,
105        drop_path_rate=0.2,
106        patch_norm=True,
107        out_indices=(0, 1, 2, 3),
108        with_cp=False,
109        convert_weights=True,
110        init_cfg=dict(type='Pretrained', checkpoint=pretrained)),
```

```
111     neck=dict(in_channels=[96, 192, 384, 768]))  
112  
113 max_epochs = 12  
114 train_cfg = dict(max_epochs=max_epochs)  
115  
116 # learning rate  
117 param_scheduler = [  
118     dict(  
119         type='LinearLR', start_factor=0.001, by_epoch=False, begin=0,  
120         end=1000),  
121     dict(  
122         type='MultiStepLR',  
123         begin=0,  
124         end=max_epochs,  
125         by_epoch=True,  
126         milestones=[8, 11],  
127         gamma=0.1)  
128 ]  
129  
130 # optimizer  
131 optim_wrapper = dict(  
132     type='OptimWrapper',  
133     paramwise_cfg=dict(  
134         custom_keys={  
135             'absolute_pos_embed': dict(decay_mult=0.),  
136             'relative_position_bias_table': dict(decay_mult=0.),  
137             'norm': dict(decay_mult=0.)  
138         }),  
139     optimizer=dict(  
140         _delete_=True,  
141         type='AdamW',  
142         lr=0.0001,  
143         betas=(0.9, 0.999),  
144         weight_decay=0.05))  
145  
146 #Entrenamiento y alteración del codigo:  
147  
148 ### Revisar que los archivos json son formato COCO  
149  
150 Importar los archivos json (entrenamiento y validación) a la siguiente ruta:  
151 - /content/detection/train/  
152 """  
153  
154 from pycocotools.coco import COCO  
155  
156 # Path to load the COCO annotation file  
157 annotation_file = '/content/detection/train/instances_train.json'  
158  
159 # Initialise the COCO object  
160 coco = COCO(annotation_file)  
161
```

```
162 # Get all category tags and corresponding category IDs
163 categories = coco.loadCats(coco.getCatIds())
164 category_id_to_name = {cat['id']: cat['name'] for cat in categories}
165
166 # Print all category IDs and corresponding category names
167 for category_id, category_name in category_id_to_name.items():
168     print(f"Category ID: {category_id}, Category Name: {category_name}")
169
170 # prompt: check if json file is in COCO format
171
172 import json
173 def cargar_json(path):
174     """
175         Cargar un archivo JSON.
176
177     Args:
178         path (str): Ruta al archivo JSON.
179
180     Returns:
181         dict: Contenido del archivo JSON
182     """
183     # Cargar el contenido del archivo JSON
184     with open(path, 'r') as file:
185         data = json.load(file)
186
187     return data
188 def is_coco_format(json_file):
189     """
190         Check if a JSON file is in COCO format.
191
192     Args:
193         json_file (str): Path to the JSON file.
194
195     Returns:
196         bool: True if the JSON file is in COCO format, False otherwise.
197     """
198
199     with open(json_file, 'r') as f:
200         data = json.load(f)
201
202     # Check if the JSON file contains the following keys:
203     required_keys = ['images', 'categories', 'annotations']
204     for key in required_keys:
205         if key not in data:
206             return False
207
208     # Check if the images key is a list of dictionaries:
209     if not isinstance(data['images'], list) or not all(isinstance(image, dict) for image in data['images']):
210         return False
211
212     # Check if the categories key is a list of dictionaries:
```

```
213     if not isinstance(data['categories'], list) or not all(isinstance(category, dict) for category in data['  
214         ↪ categories']):  
215         return False  
216  
217     # Check if the annotations key is a list of dictionaries:  
218     if not isinstance(data['annotations'], list) or not all(isinstance(annotation, dict) for annotation in data['  
219         ↪ annotations']):  
220         return False  
221  
222     return True  
223  
224     print(is_coco_format("/content/detection/train/instances_val.json"))  
225  
226     from pycocotools.coco import COCO  
227  
228     # Path to load the COCO annotation file  
229     annotation_file = '/content/detection/train/instances_val.json'  
230  
231  
232     # Initialise the COCO object  
233     coco = COCO(annotation_file)  
234  
235  
236     # Get all category tags and corresponding category IDs  
237     categories = coco.loadCats(coco.getCatIds())  
238     category_id_to_name = {cat['id']: cat['name'] for cat in categories}  
239  
240     # Print all category IDs and corresponding category names  
241     for category_id, category_name in category_id_to_name.items():  
242         print(f"Category ID: {category_id}, Category Name: {category_name}")  
243  
244     """## alteración a los archivos de configuración:  
245     subir el archivo config_swin2.py a '/content/mmdetection/configs/' y el archivo coco_instance_base a '/  
246         ↪ content/mmdetection/configs/_base_/datasets'  
247     """  
248  
249     from mmengine import Config  
250     cfg = Config.fromfile('/content/mmdetection/configs/swin/config_swin2.py')  
251  
252     from mmengine.runner import set_random_seed  
253  
254     # Modify dataset classes and color  
255     cfg.metainfo = {  
256         'classes': ('apple', ),  
257         'palette': [  
258             (220, 20, 60),  
259         ]  
260     }  
261     # Modify num classes of the model in box head and mask head  
262     cfg.model.roi_head.bbox_head.num_classes = 1  
263     cfg.model.roi_head.mask_head.num_classes = 1  
264     # Set up working dir to save files and logs.
```

```
261 cfg.work_dir = './tutorial_exps'  
262 # We can set the evaluation interval to reduce the evaluation times  
263 cfg.train_cfg.val_interval = 2  
264 # We can set the checkpoint saving interval to reduce the storage cost  
265 cfg.default_hooks.checkpoint.interval = 2  
266 # Set seed to facilitate reproducing the result  
267 cfg.seed = 0  
268 set_random_seed(0, deterministic=False)  
269 # We can also use tensorboard to log the training process  
270 cfg.visualizer.vis_backends.append({'type': 'TensorboardVisBackend'})  
271 #-----  
272 config=f'/content/mmdetection/configs/swin/config_swin2.py'  
273 with open(config, 'w') as f:  
274     f.write(cfg.pretty_text)  
275  
276 config=f'/content/mmdetection/configs/swin/config_swin2.py'  
277  
278 print(config)  
279  
280 !python tools/train.py {config}  
281  
282 # Commented out IPython magic to ensure Python compatibility.  
283 # %pip install tensorboard -i https://mirrors.ustc.edu.cn/pypi/web/simple  
284  
285 # Commented out IPython magic to ensure Python compatibility.  
286 # load tensorboard in jupyter notebook  
287 # %load_ext tensorboard  
288  
289 # Commented out IPython magic to ensure Python compatibility.  
290 # see curves in tensorboard  
291 # if you see <IPython.core.display.HTML object> please run it again  
292 # %tensorboard --logdir tutorial_exps/  
293  
294 cfg = Config.fromfile('/content/mmdetection/configs/swin/config_swin2.py')  
295  
296 import mmcv  
297 from mmdet.apis import init_detector, inference_detector  
298 img = mmcv.imread('/content/detection/train/images/20150921_132038_image286.png', channel_order='  
    ↪ rgb')  
299 checkpoint_file = '/content/mmdetection/tutorial_exps/epoch_4.pth'  
300 model = init_detector(cfg, checkpoint_file, device='cpu')  
301 new_result = inference_detector(model, img)  
302 print(new_result)  
303  
304 """.../_base_/datasets/coco_instance.py'  
305  
306 Ver los resultados  
307 """  
308  
309 from mmdet.registry import VISUALIZERS  
310 # init visualizer(run the block only once in jupyter notebook)
```

```
311 visualizer = VISUALIZERS.build(model.cfg.visualizer)
312 # the dataset_meta is loaded from the checkpoint and
313 # then pass to the model in init_detector
314 visualizer.dataset_meta = model.dataset_meta
315
316 from mmengine.visualization import Visualizer
317 # get built visualizer
318 visualizer_now = Visualizer.get_current_instance()
319 # the dataset_meta is loaded from the checkpoint and
320 # then pass to the model in init_detector
321 visualizer_now.dataset_meta = model.dataset_meta
322 # show the results
323 visualizer_now.add_datasample(
324     'new_result',
325     img,
326     data_sample=new_result,
327     draw_gt=False,
328     wait_time=0,
329     out_file=None,
330     pred_score_thr=0.1
331 )
332 visualizer_now.show()
333
334
335
336 path= '20150921_132038_image286.png'
337
338 im = mmcv.imread('/content/detection/train/images/' + path, channel_order='rgb')
339 masked = mmcv.imread('/content/detection/train/masks/' + path, channel_order='rgb')
340 def draw_mask(image, mask_generated):
341     masked_image = image.copy()
342
343     # Convert BGR image to RGB if not in RGB format
344     if image.shape[-1] == 3 and image.shape[-2] != 3:
345         masked_image = cv2.cvtColor(masked_image, cv2.COLOR_BGR2RGB)
346
347     # Create a green mask on the specified areas
348     mask = np.where(mask_generated.astype(int),
349                     np.array([0, 255, 0], dtype='uint8'),
350                     masked_image)
351
352     # Convert the masked image to uint8
353     masked_image = mask.astype(np.uint8)
354
355     # Combine the original image and the masked image with a certain weight
356     result = cv2.addWeighted(image, 0.3, masked_image, 0.7, 0)
357
358     return result
359
360 """Comparar con la imagen original con la mascara"""
361
```

```
362 cv2_imshow(draw_mask(im,masked))  
363  
364 path= 'dataset1_front_1471.png'  
365 im = mmcv.imread('/content/detection/test/images/' + path,channel_order='bgr')  
366 cv2_imshow(im)
```

7. Bibliografia

[1] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, “Swin transformer: Hierarchical vision transformer using shifted windows,” CoRR, vol. abs/2103.14030, 2021.

[2] mmdetection Respository: <https://github.com/open-mmlab/mmdetection.git>.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” CoRR, vol. abs/1706.03762, 2017.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” CoRR, vol. abs/2010.11929, 2020.

[5] N. H .ani, P. Roy, and V. Isler, “Minneapple: A benchmark dataset for apple detection and segmentation.”

[6] Swin Transformer Object Detection Repository: <https://github.com/SwinTransformer/Swin-Transformer-Object-Detection>